Mixed precision accumulation for neural network inference guided by componentwise forward error analysis

EL-MEHDI EL ARAR*
Inria, IRISA, Université de Rennes, 263 Av. Général Leclerc, F-35000, Rennes, France

SILVIU-IOAN FILIP
Inria, IRISA, Université de Rennes, 263 Av. Général Leclerc, F-35000, Rennes, France

THEO MARY
Sorbonne Université, CNRS, LIP6, 4 Place Jussieu, F-75005, Paris, France

AND

ELISA RICCIETTI
ENS de Lyon, CNRS, Inria, Université Claude Bernard Lyon 1 LIP, UMR 5668,, 69342, Lyon cedex 07, France
*Corresponding author: el-mehdi.el-arar@inria.fr

This work proposes a mathematically founded mixed precision accumulation strategy for the inference of neural networks. Our strategy is based on a new componentwise forward error analysis that explains the propagation of errors in the forward pass of neural networks. Specifically, our analysis shows that the error in each component of the output of a layer is proportional to the condition number of the inner product between the weights and the input, multiplied by the condition number of the activation function. These condition numbers can vary widely from one component to the other, thus creating a significant opportunity to introduce mixed precision: each component should be accumulated in a precision inversely proportional to the product of these condition numbers. We propose a practical algorithm that exploits this observation: it first computes all components in low precision, uses this output to estimate the condition numbers, and recomputes in higher precision only the components associated with large condition numbers. We test our algorithm on various networks and datasets and confirm experimentally that it can significantly improve the cost–accuracy tradeoff compared with uniform precision accumulation baselines.

Keywords: Neural network, inference, error analysis, mixed precision, multiply–accumulate

1. Introduction

Modern applications in artificial intelligence require increasingly complex models and thus increasing memory, time, and energy costs for storing and deploying large-scale deep learning models with parameter counts ranging in the millions and billions. This is a limiting factor both in the context of training and of inference. While the growing training costs can be tackled by the power of modern computing resources, notably GPU accelerators, the deployment of large-scale models leads to serious limitations in inference contexts with limited resources, such as embedded systems or applications that require real-time processing.

In recent years, the use of low precision arithmetic has emerged as a successful strategy to decrease these costs, motivated by the development of specialized hardware for machine learning, such as

Google's TPUs [19], NVIDIA tensor cores [1], and others [27], which provide fast mixed precision matrix multiply–accumulate (MMA) operations. Low precision is usually introduced in trained neural networks by the quantization of weights and activations, that is, by storing the network parameters in low precision [13]. Indeed, the compute workload of inference is dominated by MMA operations, which can be accelerated by using lower precisions. Quantization therefore significantly reduces the inference cost, usually in exchange for minor reductions in model accuracy. It has indeed been empirically shown that neural network inference can be done effectively even when weights and activations are stored using 8 bits [13, 24].

While weights and activations are commonly stored in low precision, the accumulation is usually done in high precision. This is partly because most specialized MMA hardware mentioned above provide the capability of accumulating in high precision with little or no performance penalty [3], and partly because accumulating in low precision can create significant numerical issues, ranging from overflow to excessive rounding error accumulation [16]. Nevertheless, reducing the accumulation precision can be an effective strategy to increase performance for general-purpose processors [9, 25, 32]. This motivates further research on how to reduce the accumulation precision as much as possible while avoiding numerical issues and preserving the model accuracy. This is the main goal of this work.

There exist several approaches to reduce the accumulation of errors in finite precision, and many of them have been considered for improving the accuracy of the training and/or the inference. For example, stochastic rounding [7, 8] prevents errors from accumulating all in the same direction and thus improves the average accuracy; it has been used to accelerate training [12, 15]. Blocked summation methods [16, Chap. 4], [4] reduce the worst-case error bounds by constraining the summation order, and has also been used for training acceleration [31]. Scaling techniques can help to avoid overflow and minimize underflow [21], and have especially received focus in the context of fixed-point arithmetic [5, 6, 25, 26, 32].

All previously mentioned works only consider *uniform precision* accumulation, that is, the accumulation precision is the same across all inner products (multiply–accumulate operations). In this work, we will instead focus on *mixed precision* accumulation, that is, we will allow different inner products to be performed in different precisions. The main advantage of mixed precision approaches is that they can leverage the possible differences in sensitivity of different parts of the computation: whereas a uniform precision scheme would be limited by the most sensitive parts that require the highest precision, a mixed precision scheme can adaptively keep only these parts in high precision, while switching the less sensitive parts to lower precision—ideally without (significantly) impacting the model accuracy.

While mixed precision approaches have been extensively investigated for *quantization* [10, 11, 14, 20, 29, 30, 33, 34], to the best of our knowledge, they have not been previously considered for *accumulation*. This work is therefore completely complementary to existing studies. On the one hand, our approach is agnostic with respect to the quantization method (that is, it applies to any network, regardless of how it has been quantized). On the other hand, it considers two different accumulation precisions with unit roundoffs u_{low} and u_{high}, but does not otherwise make any specific assumptions on how this accumulation is performed: that is, our approach may be combined with stochastic rounding, blocked summation, etc.; the specific choice of accumulation method will simply determine just how low u_{low} can be, and how high u_{high} needs to be.

The key question that our work addresses is: how should we decide which inner products to perform in which precision? Our approach aims at answering this question in a mathematically founded way by basing the precision choice criterion on a rigorous error analysis. We develop such an analysis that considers an inexact inference with a very generic error model. Our analysis is in spirit quite similar to

the recent work of Beuzeville et al. [2], which also analyzes the inference of neural networks in presence of errors. However, there are some key differences between the two analyses. Indeed, Beuzeville et al. perform a *backward* error analysis, whereas we will focus on the *forward* error. There are advantages to both types of analyses: backward error analysis yields bounds that are mostly independent of the neural network parameters (they depend on the number and size of the layers, but not on the actual values of the weights), and allows for establishing the numerical stability of inference—the main goal and result of [2]. In contrast, the goal of our forward error analysis is completely different: we seek bounds that directly relate the errors incurred in each inner product to the accuracy of the final output of the network, in order to identify possible mixed precision opportunities; thus, our bounds strongly depend on the network parameter values, and this is precisely what we exploit to develop a mixed precision strategy. Most importantly, the analysis of Beuzeville et al. bounds the *normwise* error, that is, the error is only bounded in (some) norm; this does not allow to distinguish the impact of errors incurred in different components of each layer of the network: the errors across different components are "smudged" together in norm. In contrast, our analysis bounds the *componentwise* error; this allows us to precisely identify the size of the errors in each component. In particular, we make the key observation that the error incurred in each component is proportional to both the condition number of the inner product and the condition number of the activation function evaluated at that component. In order to balance the errors across all components, we should therefore set the precision of each inner product to be inversely proportional to the associated condition number. Because the magnitude of these condition numbers can vary widely from one component to the other, this creates a significant opportunity for mixed precision.

To summarize, the first main contribution of this work is to perform a componentwise forward error analysis that guides us towards a mixed precision inference evaluation strategy. The second main contribution of this work is to develop a practical mixed precision algorithm that is guided by this analysis. In order to make the algorithm practical, we must introduce some approximations: computing the exact condition numbers would indeed be too expensive. Motivated by some empirical observations, we however show that the condition numbers can be cheaply estimated as a by-product of the output of each layer computed in low precision. Therefore, we propose the following approach, summarized in Figure 1.1: at each layer ℓ, we first compute the output $h_\ell = \phi_\ell(W_\ell h_{\ell-1})$ entirely (uniformly) in a low precision u_{low}. Then, we estimate the condition number κ_ℓ and check each of its components $(\kappa_\ell)_i$: components for which the condition number is small enough ($(\kappa_\ell)_i \leq \tau$, for some tolerance τ) are kept in precision u_{low}, whereas those for which the condition number is too large ($(\kappa_\ell)_i > \tau$) are recomputed using a higher precision u_{high}.

We test the proposed algorithm on multilayer perceptrons networks of various depth, trained on the MNIST and Fashion MNIST datasets. Our experiments show that the algorithm can achieve a flexible cost–accuracy tradeoff, tunable via the tolerance parameter τ. Crucially, the achieved tradeoff is in many cases significantly better than with uniform precision accumulation: that is, our mixed precision accumulation approach can significantly improve the model accuracy compared with a uniform low precision approach, for a significantly lower cost than the uniform high precision approach.

The rest of the paper is organized as follows: in section 2 we carry out our error analysis and discuss its significance. In section 3, we develop an inference algorithm with mixed precision accumulation. We test the algorithm experimentally in section 4. Finally, we conclude in section 5.
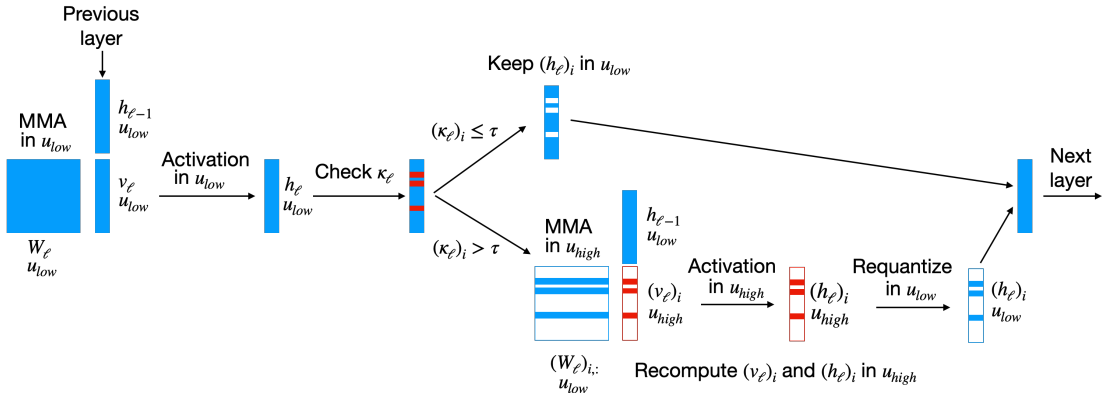
FIG. 1.1. *Illustration of our inference approach with mixed precision accumulation (Algorithm 3.1). At each layer ℓ we first compute the MMA $v_\ell = W_\ell h_{\ell-1}$ (where $h_{\ell-1}$ is the output of the previous layer) and the activation $h_\ell = \phi_\ell(v_\ell)$ (where ϕ_ℓ is the activation function) in uniform low precision u_{low}. We estimate the condition number κ_ℓ and use it to decide which components can be kept in low precision (those for which $(\kappa_\ell)_i \leq \tau$, for some tolerance τ) and which must be recomputed in higher precision u_{high}; the latter are then requantized to low precision and recombined with the components kept in low precision to produce the final output of the layer, which is passed to the next layer.*

2. Componentwise error analysis

2.1. *Setting, notations, and error model*

We consider feedforward networks with L layers, where each layer is indexed by $\ell = 1, \ldots, L$ and composed of n_ℓ neurons. We denote by $W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ the matrices of weights and by $\phi_\ell : \mathbb{R} \mapsto \mathbb{R}$ the activation functions applied componentwise. For an input $x \in \mathbb{R}^{n_0}$, we denote $h_0 = x$ and for each layer ℓ, the output of the layer $h_\ell \in \mathbb{R}^{n_\ell}$ is computed as

$$h_\ell = \phi_\ell(W_\ell h_{\ell-1}).$$

While we do use bias terms in the experiments in section 4, we do not include them explicitly in the presented analysis for simplicity. The bias terms b_ℓ could be easily included by redefining the weight matrices as $W'_\ell = \begin{bmatrix} W_\ell & b_\ell \end{bmatrix}$ and the output of the $(\ell-1)$th layer as $h'_{\ell-1} = \begin{bmatrix} h_{\ell-1} & 1 \end{bmatrix}^T$. We then have

$$W_\ell h_{\ell-1} + b_\ell = \begin{bmatrix} W_\ell & b_\ell \end{bmatrix} \begin{bmatrix} h_{\ell-1} \\ 1 \end{bmatrix} = W'_\ell h'_{\ell-1}.$$

We will use the following notations. Quantities affected by an error are marked by a hat. We denote by $\circ$ the Hadamard (componentwise) product and by $\oslash$ the Hadamard division; the Hadamard product of a matrix with a vector multiplies the rows of the matrix by the components of the vector. We denote by $|\cdot|$ the absolute value, which is applied componentwise for vectors and matrices. Inequalities between vectors $x \leq y$ or matrices $A \leq B$ of identical dimensions also apply componentwise; moreover, an inequality $A \leq x$ between a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $x \in \mathbb{R}^m$ applies to each row of A componentwise, that is, $a_{ij} \leq x_i$ for all i, j. We denote by $\mathbf{1}$ the matrix or vector of all ones.

We seek to analyze the effect of errors in the computation of h_ℓ. To do so, we will use the following generic error model.

Model 2.1   *We assume that* $\widehat{h}_0 = h_0 = x$ *and that each computed* $\widehat{h}_\ell$ *satisfies*

$$\widehat{h}_\ell = \phi_\ell\big((W_\ell \circ (\mathbf{1} + \Delta W_\ell))\widehat{h}_{\ell-1}\big) \circ (\mathbf{1} + \Delta\phi_\ell), \quad |\Delta W_\ell| \le \varepsilon_\ell^W, \quad |\Delta\phi_\ell| \le \varepsilon_\ell^\phi, \tag{2.1}$$

where $\Delta W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$, $\Delta\phi_\ell \in \mathbb{R}^{n_\ell}$, $\varepsilon_\ell^W \in \mathbb{R}^{n_\ell}$ *is a nonnegative vector whose components bound the backward errors incurred in the evaluation of the matrix–vector product with* W_ℓ, *so that* $(\varepsilon_\ell^W)_i = \max_{1 \le j \le n_{\ell-1}} |(\Delta W_\ell)_{ij}|$ *for* $i = 1, \ldots, n_\ell$, *and* $\varepsilon_\ell^\phi \in \mathbb{R}^{n_\ell}$ *is a nonnegative vector whose components bound the forward errors incurred in the evaluation of* ϕ_ℓ.

2.2. Preliminaries

We will need the following two inequalities on perturbed matrix–vector products.

Lemma 2.2   *Let* $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, *and* $\Delta x \in \mathbb{R}^n$. *We have*

$$|A||x \circ \Delta x| \le \|\Delta x\|_\infty |A||x|. \tag{2.2}$$

Proof Since the inequality is componentwise, it suffices to prove it for an arbitrary index i, $1 \le i \le m$. The ith component of $|A||x \circ \Delta x|$ satisfies

$$(|A||x \circ \Delta x|)_i = \sum_{j=1}^n |a_{ij} x_j \Delta x_j| \le \|\Delta x\|_\infty \sum_{j=1}^n |a_{ij}||x_j| = \|\Delta x\|_\infty (|A||x|)_i. \qquad \square$$

Lemma 2.3   *Let* $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, *and* $\Delta A \in \mathbb{R}^{m \times n}$ *such that* $|\Delta A| \le \varepsilon^A \in \mathbb{R}^m$ *with* $(\varepsilon^A)_i = \max_{1 \le j \le n} |\Delta a_{ij}|$. *We have*

$$|A \circ \Delta A||x| \le (|A||x|) \circ \varepsilon^A. \tag{2.3}$$

Proof Once again, since the inequality is componentwise, it suffices to prove it for an arbitrary index i, $1 \le i \le m$. The ith component of $|A \circ \Delta A||x|$ satisfies

$$(|A \circ \Delta A||x|)_i = \sum_{j=1}^n |a_{ij} \Delta a_{ij} x_j| \le \sum_{j=1}^n |a_{ij}||x_j|(\varepsilon^A)_i = ((|A||x|) \circ \varepsilon^A)_i. \qquad \square$$

Lemma 2.2 states that multiplying a nonnegative matrix $|A|$ with a nonnegative vector $|x|$ perturbed componentwise by $|\Delta x|$ yields a result $|A||x|$ whose ith component is perturbed by the largest of the components of $|\Delta x|$. Lemma 2.3 shows that a similar result holds when multiplying a nonnegative matrix $|A|$ perturbed componentwise by $|\Delta A|$ with a nonnegative vector $|x|$: this yields a result $|A||x|$ whose ith component is perturbed by the largest of the components of the ith row of $|\Delta A|$, $(\varepsilon^A)_i = \max_{1 \le j \le n} |\Delta a_{ij}|$. In other words, perturbed matrix–vectors (with a componentwise perturbation on either the matrix or the vector) contaminate the result by spreading the perturbation across its components.

We next define two key quantities that will appear in the analysis: the condition numbers of a matrix–vector product and of a function.

Condition number of a matrix–vector product. For $A \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$, we have

$$|A||x| = \kappa_{A,x} \circ |Ax|, \tag{2.4}$$

where $\kappa_{A,x} \in \mathbb{R}^m$ is the vector whose ith component

$$(\kappa_{A,x})_i = \frac{(|A||x|)_i}{|Ax|_i} \tag{2.5}$$

is the condition number of the dot product between the ith row of A and x, which reflects the possibility of cancellation [16, sect. 1.7] in the computation of Ax when $|A||x| > |Ax|$.

Condition number of a function. During the evaluation of $\phi_\ell(v)$ for some vector $v \in \mathbb{R}^{n_\ell}$, we will also need to express a relative perturbation Δv on the input v as a relative perturbation $\Delta \phi_\ell(v)$ on the output $\phi_\ell(v)$. To do so, we introduce a function $\kappa_{\phi_\ell} : \mathbb{R}^{n_\ell} \mapsto \mathbb{R}_+^{n_\ell}$ that satisfies

$$\phi_\ell\big(v \circ (\mathbf{1} + \Delta v)\big) = \phi_\ell(v) \circ \big(\mathbf{1} + \kappa_{\phi_\ell}(v) \circ \Delta v'\big), \quad \Delta v' = \pm \Delta v. \tag{2.6}$$

Equality (2.6) is stating that a relative perturbation Δv on the input v leads to a relative perturbation on the output $\phi_\ell(v)$ of magnitude $\kappa_{\phi_\ell}(v)|\Delta v|$ (note that we introduce a perturbation $\Delta v'$ to account for a possible change of sign).

To obtain a more explicit expression of κ_ϕ, consider the case where $v \in \mathbb{R}$. Then (2.6) becomes $\phi_\ell(v(1 + \Delta v)) = \phi_\ell(v)(1 + \kappa_{\phi_\ell}(v)\Delta v)$. Assuming first that $\phi_\ell(v)\Delta v \neq 0$, this yields the expression

$$\kappa_{\phi_\ell}(v) = \frac{|\phi_\ell(v(1 + \Delta v)) - \phi_\ell(v)|}{|\phi_\ell(v)\Delta v|}. \tag{2.7}$$

Taking the limit as Δv goes to zero gives the condition number of ϕ_ℓ at v, $|v\phi_\ell'(v)/\phi_\ell(v)|$ [16, sect. 1.8], which shows that κ_{ϕ_ℓ} can be interpreted as the condition number of ϕ_ℓ for small perturbations. The case where $\phi_\ell(v)\Delta v = 0$ requires special care. If $\Delta v = 0$, or if $\phi_\ell(v) = \phi_\ell(v(1 + \Delta v)) = 0$, then (2.6) is satisfied for any κ_{ϕ_ℓ}, so we may in particular define $\kappa_{\phi_\ell} = 0$. If $\phi_\ell(v) = 0$ but $\phi_\ell(v(1 + \Delta v)) \neq 0$, then there does not exist any finite κ_{ϕ_ℓ} such that (2.6) is satisfied, and so we define $\kappa_{\phi_\ell} = \infty$. To summarize, we have the explicit expression of κ_{ϕ_ℓ}

$$\kappa_{\phi_\ell}(v) = \begin{cases} \frac{|\phi_\ell(v(1+\Delta v)) - \phi_\ell(v)|}{|\phi_\ell(v)\Delta v|} & \text{if } \phi_\ell(v)\Delta v \neq 0 \\ 0 & \text{if } \phi_\ell(v) = \phi_\ell(v(1+\Delta v)) \\ \infty & \text{if } \phi_\ell(v) = 0 \text{ and } \phi_\ell(v(1+\Delta v)) \neq 0. \end{cases} \tag{2.8}$$

Note that the fact that κ_{ϕ_ℓ} can take ∞ as a value is largely an artifact of considering relative perturbations. For example, for ReLU activation, $\kappa_{\phi_\ell}(v) = \infty$ occurs only when we simultaneously have $v < 0$ and $v(1 + \Delta v) > 0$. These conditions are met when $\Delta v < -1$, which corresponds to a relative error $|\Delta v|$ greater than 1.

Going back to the general case where ϕ_ℓ takes $v \in \mathbb{R}^{n_\ell}$ as input, since (2.6) is a componentwise definition, we obtain the expression of the ith component of $\kappa_{\phi_\ell}(v)$ by applying (2.8) to $\kappa_{\phi_\ell}(v_i)$.

2.3. *The analysis*

We are now ready to analyze the computation of h_ℓ. We proceed by induction: assuming that the computed $\widehat{h}_{\ell-1}$ satisfies

$$\widehat{h}_{\ell-1} = h_{\ell-1} \circ (\mathbf{1} + \Delta h_{\ell-1}), \quad |\Delta h_{\ell-1}| \le \varepsilon_{\ell-1}^h \in \mathbb{R}^{n_{\ell-1}} \tag{2.9}$$

for some error term $\Delta h_{\ell-1}$ bounded componentwise by $\varepsilon_{\ell-1}^h$, we seek to determine Δh_ℓ and its corresponding bound ε_ℓ^h. Defining $v_\ell = W_\ell h_{\ell-1}$ and injecting (2.9) into (2.1), we obtain

$$
\begin{aligned}
\widehat{h}_\ell &= \phi_\ell \Big(\big(W_\ell \circ (\mathbf{1} + \Delta W_\ell) \big) \big(h_{\ell-1} \circ (\mathbf{1} + \Delta h_{\ell-1}) \big) \Big) \circ (\mathbf{1} + \Delta \phi_\ell) \\
&= \phi_\ell \Big(v_\ell + (W_\ell \circ \Delta W_\ell) h_{\ell-1} + W_\ell (h_{\ell-1} \circ \Delta h_{\ell-1}) + (W_\ell \circ \Delta W_\ell)(h_{\ell-1} \circ \Delta h_{\ell-1}) \Big) \circ (\mathbf{1} + \Delta \phi_\ell) \\
&= \phi_\ell \big(v_\ell \circ (\mathbf{1} + \Delta v_\ell) \big) \circ (\mathbf{1} + \Delta \phi_\ell),
\end{aligned} \tag{2.10}
$$

with

$$|\Delta v_\ell| \le \Big(|(W_\ell \circ \Delta W_\ell) h_{\ell-1}| + |W_\ell (h_{\ell-1} \circ \Delta h_{\ell-1})| + |(W_\ell \circ \Delta W_\ell)(h_{\ell-1} \circ \Delta h_{\ell-1})| \Big) \oslash |v_\ell|. \tag{2.11}$$

Using Lemmas 2.2 and 2.3 together with (2.1) and (2.9), we have

$$|\Delta v_\ell| \le (|W_\ell||h_{\ell-1}|) \circ \varepsilon_\ell^W \oslash |v_\ell| + \|\varepsilon_{\ell-1}^h\|_\infty (|W_\ell||h_{\ell-1}|) \oslash |v_\ell| + \|\varepsilon_{\ell-1}^h\|_\infty (|W_\ell||h_{\ell-1}|) \circ \varepsilon_\ell^W \oslash |v_\ell|. \tag{2.12}$$

By (2.4) we have

$$(|W_\ell||h_{\ell-1}|) \oslash |v_\ell| = \kappa_{W_\ell, h_{\ell-1}} =: \kappa_{v_\ell}, \tag{2.13}$$

where, for the sake of readability, we abbreviate $\kappa_{W_\ell, h_{\ell-1}}$ as κ_{v_ℓ}. We thus obtain

$$
\begin{aligned}
|\Delta v_\ell| &\le \kappa_{v_\ell} \circ \big(\varepsilon_\ell^W + \|\varepsilon_{\ell-1}^h\|_\infty \mathbf{1} + \|\varepsilon_{\ell-1}^h\|_\infty \varepsilon_\ell^W \big) \\
&= \kappa_{v_\ell} \circ \big(\varepsilon_\ell^W + \|\varepsilon_{\ell-1}^h\|_\infty (\mathbf{1} + \varepsilon_\ell^W) \big).
\end{aligned} \tag{2.14}
$$

Using (2.6) in (2.10), we have

$$
\begin{aligned}
\widehat{h}_\ell &= \phi_\ell(v_\ell) \circ (\mathbf{1} + \kappa_{\phi_\ell}(v_\ell) \circ \pm \Delta v_\ell) \circ (\mathbf{1} + \Delta \phi_\ell) \\
&= h_\ell \circ (\mathbf{1} + \kappa_{\phi_\ell}(v_\ell) \circ \pm \Delta v_\ell) \circ (\mathbf{1} + \Delta \phi_\ell) \\
&= h_\ell \circ (\mathbf{1} + \kappa_{\phi_\ell}(v_\ell) \circ \pm \Delta v_\ell + \Delta \phi_\ell + \kappa_{\phi_\ell}(v_\ell) \circ \pm \Delta v_\ell \circ \Delta \phi_\ell) \\
&= h_\ell \circ (\mathbf{1} + \Delta h_\ell)
\end{aligned}
$$

with

$$
\begin{aligned}
|\Delta h_\ell| &\le \kappa_{\phi_\ell}(v_\ell) \circ |\Delta v_\ell| + |\Delta \phi_\ell| + \kappa_{\phi_\ell}(v_\ell) \circ |\Delta v_\ell| \circ |\Delta \phi_\ell| \\
&= \kappa_{\phi_\ell}(v_\ell) \circ |\Delta v_\ell| \circ (\mathbf{1} + |\Delta \phi_\ell|) + |\Delta \phi_\ell|
\end{aligned} \tag{2.15}
$$

Combining (2.1) and (2.14) into (2.15), we finally obtain

$$|\Delta h_\ell| \le \kappa_{\phi_\ell}(v_\ell) \circ \kappa_{v_\ell} \circ \big(\varepsilon_\ell^W + \|\varepsilon_{\ell-1}^h\|_\infty (\mathbf{1} + \varepsilon_\ell^W) \big) \circ (\mathbf{1} + \varepsilon_\ell^\phi) + \varepsilon_\ell^\phi =: \varepsilon_\ell^h. \tag{2.16}$$

We summarize our analysis in the following theorem.

Theorem 2.4 *Let $h_\ell = \phi_\ell(W_\ell h_{\ell-1})$ be computed inexactly such that the computed $\widehat{h}_\ell$ satisfies Model 2.1. Then, we have*

$$\widehat{h}_\ell = h_\ell \circ (\mathbf{1} + \Delta h_\ell), \quad |\Delta h_\ell| \leq \varepsilon_\ell^h,$$

where ε_ℓ^h satisfies the recurrence relation

$$\varepsilon_\ell^h = \kappa_{\phi_\ell}(v_\ell) \circ \kappa_{v_\ell} \circ \left(\varepsilon_\ell^W + \|\varepsilon_{\ell-1}^h\|_\infty (\mathbf{1} + \varepsilon_\ell^W)\right) \circ (\mathbf{1} + \varepsilon_\ell^\phi) + \varepsilon_\ell^\phi,$$

where κ_{ϕ_ℓ} satisfies (2.6), κ_{v_ℓ} is defined in (2.13), and $\varepsilon_{\ell-1}^h$ bounds the relative error incurred in the computation of $h_{\ell-1}$ as defined in (2.9).

2.4. Interpretation of the analysis and consequences

We now explain why this analysis reveals important features of the behavior of the forward propagation under error perturbations, and motivates the use of mixed precision. Theorem 2.4 shows that, to first order, we have the recurrence

$$\varepsilon_\ell^h = \kappa_{\phi_\ell}(v_\ell) \circ \kappa_{v_\ell} \circ (\varepsilon_\ell^W + \|\varepsilon_{\ell-1}^h\|_\infty \mathbf{1}) + \varepsilon_\ell^\phi. \tag{2.17}$$

This means that at layer ℓ, the previously accumulated error $\varepsilon_{\ell-1}^h$ undergoes a series of transformations due to the propagation process. First, we add the local backward error ε_ℓ^W accounting for the inexact matrix–vector product. Then, the combined error is scaled componentwise by the condition numbers $\kappa_{\phi_\ell}(v_\ell)$ and κ_{v_ℓ}, which quantify the sensitivity of the layer's operations to input perturbations. This scaling reflects how the local structure of the layer amplifies the existing errors. Finally, we add the error ε_ℓ^ϕ accounting for the inexact evaluation of the activation function.

We can derive from recurrence (2.17) a simpler scalar recurrence on $\|\varepsilon_\ell^h\|_\infty$:

$$\|\varepsilon_\ell^h\|_\infty = \|\kappa_{\phi_\ell}(v_\ell) \circ \kappa_{v_\ell} \circ \varepsilon_\ell^W\|_\infty + \|\kappa_{\phi_\ell}(v_\ell) \circ \kappa_{v_\ell}\|_\infty \|\varepsilon_{\ell-1}^h\|_\infty + \|\varepsilon_\ell^\phi\|_\infty. \tag{2.18}$$

This yields the following corollary.

Corollary 2.5 *For all $\ell = 1, \ldots, L$, let*

$$\widehat{h}_\ell = h_\ell \circ (\mathbf{1} + \Delta h_\ell), \quad |\Delta h_\ell| \leq \varepsilon_\ell^h,$$

and assume ε_ℓ^h satisfies the recurrence relation (2.18). Then the computed final output of the network, $\widehat{h}_L$, satisfies

$$\widehat{h}_L = h_L \circ (\mathbf{1} + \Delta h_L), \quad |\Delta h_L| \leq \varepsilon_L^h,$$

with

$$\|\varepsilon_L^h\|_\infty = \sum_{\ell=1}^{L} \left[\left(\prod_{k=\ell+1}^{L} \|\kappa_{\phi_k}(v_k) \circ \kappa_{v_k}\|_\infty \right) \left(\|\kappa_{\phi_\ell}(v_\ell) \circ \kappa_{v_\ell} \circ \varepsilon_\ell^W\|_\infty + \|\varepsilon_\ell^\phi\|_\infty \right) \right]. \tag{2.19}$$

Proof The proof is by induction on L. For $L = 1$, using (2.18) gives

$$\|\varepsilon_1^h\|_\infty = \|\kappa_{\phi_1}(v_1) \circ \kappa_{v_1} \circ \varepsilon_1^W\|_\infty + \|\kappa_{\phi_1}(v_1) \circ \kappa_{v_1}\|_\infty \|\varepsilon_0^h\|_\infty + \|\varepsilon_1^\phi\|_\infty.$$

Since $\widehat{h}_0 = h_0$, ε_0^h is zero and (2.19) holds for $L = 1$. For the inductive step, assume that (2.19) is true for $L - 1$. By (2.18) we have

$$\|\varepsilon_L^h\|_\infty = \|\kappa_{\phi_L}(v_L) \circ \kappa_{v_L} \circ \varepsilon_L^W\|_\infty + \|\kappa_{\phi_L}(v_L) \circ \kappa_{v_L}\|_\infty \|\varepsilon_{L-1}^h\|_\infty + \|\varepsilon_L^\phi\|_\infty$$

and by the inductive assumption we thus obtain

$$
\begin{aligned}
\|\varepsilon_L^h\|_\infty &= \|\kappa_{\phi_L}(v_L) \circ \kappa_{v_L} \circ \varepsilon_L^W\|_\infty + \|\varepsilon_L^\phi\|_\infty \\
&\quad + \|\kappa_{\phi_L}(v_L) \circ \kappa_{v_L}\|_\infty \sum_{\ell=1}^{L-1} \left[\left(\prod_{k=\ell+1}^{L-1} \|\kappa_{\phi_k}(v_k) \circ \kappa_{v_k}\|_\infty \right) \left(\|\kappa_{\phi_\ell}(v_\ell) \circ \kappa_{v_\ell} \circ \varepsilon_\ell^W\|_\infty + \|\varepsilon_\ell^\phi\|_\infty \right) \right] \\
&= \|\kappa_{\phi_L}(v_L) \circ \kappa_{v_L} \circ \varepsilon_L^W\|_\infty + \|\varepsilon_L^\phi\|_\infty \\
&\quad + \sum_{\ell=1}^{L-1} \left[\left(\prod_{k=\ell+1}^{L} \|\kappa_{\phi_k}(v_k) \circ \kappa_{v_k}\|_\infty \right) \left(\|\kappa_{\phi_\ell}(v_\ell) \circ \kappa_{v_\ell} \circ \varepsilon_\ell^W\|_\infty + \|\varepsilon_\ell^\phi\|_\infty \right) \right] \\
&= \sum_{\ell=1}^{L} \left[\left(\prod_{k=\ell+1}^{L} \|\kappa_{\phi_k}(v_k) \circ \kappa_{v_k}\|_\infty \right) \left(\|\kappa_{\phi_\ell}(v_\ell) \circ \kappa_{v_\ell} \circ \varepsilon_\ell^W\|_\infty + \|\varepsilon_\ell^\phi\|_\infty \right) \right]. \qquad \square
\end{aligned}
$$

Minimizing the error bound $\|\varepsilon_L^h\|_\infty$ on the final output of the network thus amounts to minimizing each of the error terms in sum (2.19). Assuming that the input x and the weights of the network W_ℓ are fixed, the only quantities under our control in this expression are ε_ℓ^W and ε_ℓ^ϕ, that is, the precision at which we evaluate the matrix–vector products and the activation functions. We are interested in using the lowest possible precisions while still achieving an error under a given accuracy target: $\|\varepsilon_L^h\|_\infty \leq \varepsilon$. To do so, it seems sensible to equilibrate as much as possible the errors on each of the terms in (2.19), that is,

$$\left(\prod_{k=\ell+1}^{L} \|\kappa_{\phi_k}(v_k) \circ \kappa_{v_k}\|_\infty \right) \left(\|\kappa_{\phi_\ell}(v_\ell) \circ \kappa_{v_\ell} \circ \varepsilon_\ell^W\|_\infty + \|\varepsilon_\ell^\phi\|_\infty \right) \leq \varepsilon/L. \tag{2.20}$$

Equation (2.20) shows that the errors incurred at layer ℓ are multiplied by the condition numbers of all the successive layers, $\prod_{k=\ell+1}^{L} \|\kappa_{\phi_k}(v_k) \circ \kappa_{v_k}\|_\infty$. In principle, this quantity may vary across layers (in fact, it decreases monotonically as ℓ increases). However, because the errors are taken in infinity norm, only the maximum error components of subsequent layers play a role: the potential variations across components are smudged together. Since this term is moreover not easy to compute or estimate in practice, it seems reasonable to ignore it and rather consider the following criterion:

$$\|\kappa_{\phi_\ell}(v_\ell) \circ \kappa_{v_\ell} \circ \varepsilon_\ell^W\|_\infty + \|\varepsilon_\ell^\phi\|_\infty \leq \varepsilon/L. \tag{2.21}$$

From this, we can immediately notice that the errors ε_ℓ^ϕ from the activation functions appear in the infinity norm. This suggests that it is meaningless to vary the precision of the activations between different components, because only the maximum error component from the previous layer is propagated; thus we may as well compute all the components in the same precision.
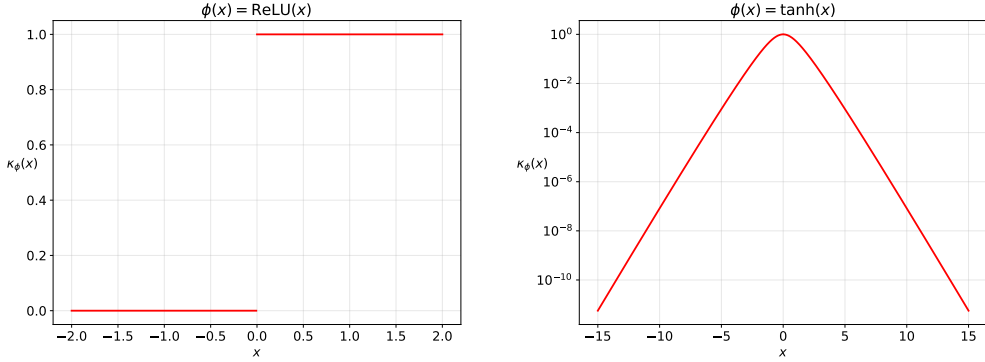
FIG. 2.1. *Condition number* $\kappa_\phi(x) = \frac{|\phi'(x)x|}{|\phi(x)|}$ *for* $\phi(x) = \mathrm{ReLU}(x)$ *(left) and* $\phi(x) = \tanh(x)$ *(right).*

On the other hand, ε_ℓ^W is multiplied componentwise by the condition number

$$\kappa_\ell := \kappa_{\phi_\ell}(v_\ell) \circ \kappa_{v_\ell}, \tag{2.22}$$

so we should try to balance each component of $\kappa_\ell \circ \varepsilon_\ell^W$ to minimize their maximum. Therefore, we should choose the precision of the inner product with the ith row of W_ℓ to be inversely proportional to the ith component of κ_ℓ. This represents a good opportunity to introduce mixed precision in the forward pass: we expect the components of κ_ℓ to have a large dynamic range. Indeed, for typical activation functions such as ReLU or tanh, Figure 2.1 shows that $\kappa_{\phi_\ell} \leq 1$, and some of its components may be much smaller than 1, meaning that some inner products can be computed in very low precision, while still maintaining a high accuracy on the overall computation.

In the next section we develop a mixed precision algorithm based on this reasoning.

3. A mixed precision algorithm for NN inference

In this section, we show how to exploit the analysis presented in the previous section to introduce mixed precision in the feedforward pass of neural networks. We assume to have a trained network with given floating-point weights W_ℓ, $\ell = 1, \ldots, L$ stored in precision u_{low}, and we seek to exploit mixed precision in the computation of the output of the network for a given input x.

3.1. *Main principle*

As discussed in the previous section, the errors at layer ℓ are proportional to the product $\kappa_\ell \circ \varepsilon_\ell^W$ (see (2.21)) and our objective is to balance each component so as to minimize the maximum $\|\kappa_\ell \circ \varepsilon_\ell^W\|_\infty$. Ideally, if the condition numbers κ_ℓ were readily available, the precisions of each inner product would simply be chosen such that $(\varepsilon_\ell^W)_i \leq \varepsilon/(\kappa_\ell)_i$, for a given target accuracy $\varepsilon > 0$; this choice would indeed yield $\|\kappa_\ell \circ \varepsilon_\ell^W\|_\infty \leq \varepsilon$. This shows that the precision used to compute each component of the ℓth layer should be chosen to be inversely proportional to the corresponding component of the condition number κ_ℓ. Let us consider the use of two precisions, with unit roundoffs $u_{\text{high}} < u_{\text{low}}$. Then for large components of κ_ℓ we should be careful in using the high precision u_{high}, whereas for small components, the errors incurred will be damped and so we can safely use the lower precision u_{low} without impacting the accuracy of the output. Concretely, we can introduce a tolerance $\tau > 0$ which controls the precision
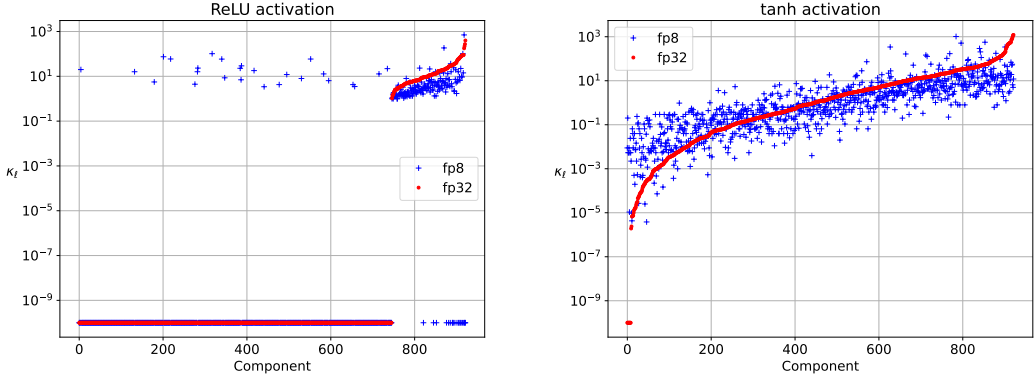
FIG. 3.1. *Comparison of the condition numbers $\kappa_\ell = \kappa_\phi \circ \kappa_{v_\ell}$ depending on whether they are computed in FP32 or in FP8, for a 3-layer network trained on the MNIST dataset with ReLU (left) and tanh (right) activations. The values are sorted with respect to the FP32 condition numbers.*

switch criterion: if $(\kappa_\ell)_i \leq \tau$ we use precision u_{low}, otherwise we use precision u_{high}. In particular, if the inner product between the ith row of W_ℓ and $h_{\ell-1}$ is implemented in floating-point arithmetic with a unit roundoff u_i (equal to either u_{low} or u_{high}), rounding error analysis [18] shows that $(\varepsilon_\ell^W)_i = n_{\ell-1} u_i$. Thus, in order for $\|\kappa_\ell \circ \varepsilon_\ell^W\|_\infty \leq \varepsilon$ to hold, we should set the tolerance as $\tau = \varepsilon/(n_{\ell-1} u_{\text{low}})$.

3.2. *From a theoretical to a practical criterion: estimating κ_ℓ*

While the principle behind this strategy would be mathematically ideal, unfortunately, since we do not know the values of κ_ℓ, it cannot be implemented as it is in practice. Indeed, it is worth recalling that $\kappa_\ell = \kappa_{\phi_\ell}(v_\ell) \circ \kappa_{v_\ell}$ depends on $v_\ell = W_\ell h_{\ell-1}$; therefore, computing κ_ℓ and thus v_ℓ in high precision would defeat the purpose of using mixed precision, since v_ℓ is precisely the result of the matrix–vector product that we aim to accelerate. Moreover, for any layer ℓ, v_ℓ depends in particular on $h_0 = x$, the input of the network, so the precision choices depend on the input and cannot be reused across different inputs.

In order to obtain a practical algorithm, we introduce some approximations. The key observation is that we do not need a very accurate computation of κ_ℓ: estimating its order of magnitude is sufficient to decide which precision to use. Therefore, this suggests the following idea: for each layer, compute first v_ℓ in precision u_{low}, that is, perform the entire matrix–vector product in low precision. Then, use this approximate v_ℓ to compute an estimated κ_ℓ and check the criterion for each component $(\kappa_\ell)_i$: if $(\kappa_\ell)_i \leq \tau$, the component $(v_\ell)_i$ computed in low precision can be kept, whereas if $(\kappa_\ell)_i > \tau$, $(v_\ell)_i$ should be recomputed in high precision u_{high}. This approach will therefore work best in situations where most components can be computed in low precision, and high precision is only needed to recompute a few of the most sensitive components. Indeed, if the criterion leads to too many components needing to be recomputed, this mixed precision approach may end up being more expensive than simply computing everything in high precision from the start.

To assess whether computing κ_ℓ in low precision is a reasonable approximation in practice, we compare in Figure 3.1 the values of the condition numbers computed in FP32 (red) with the corresponding values computed in FP8 (blue). We use a three-layer perceptron network trained on the MNIST dataset for the ReLU (left plot) and tanh (right plot) activation functions. The figure shows that the values computed in FP8 follow the same trend as those computed in FP32, thus providing a
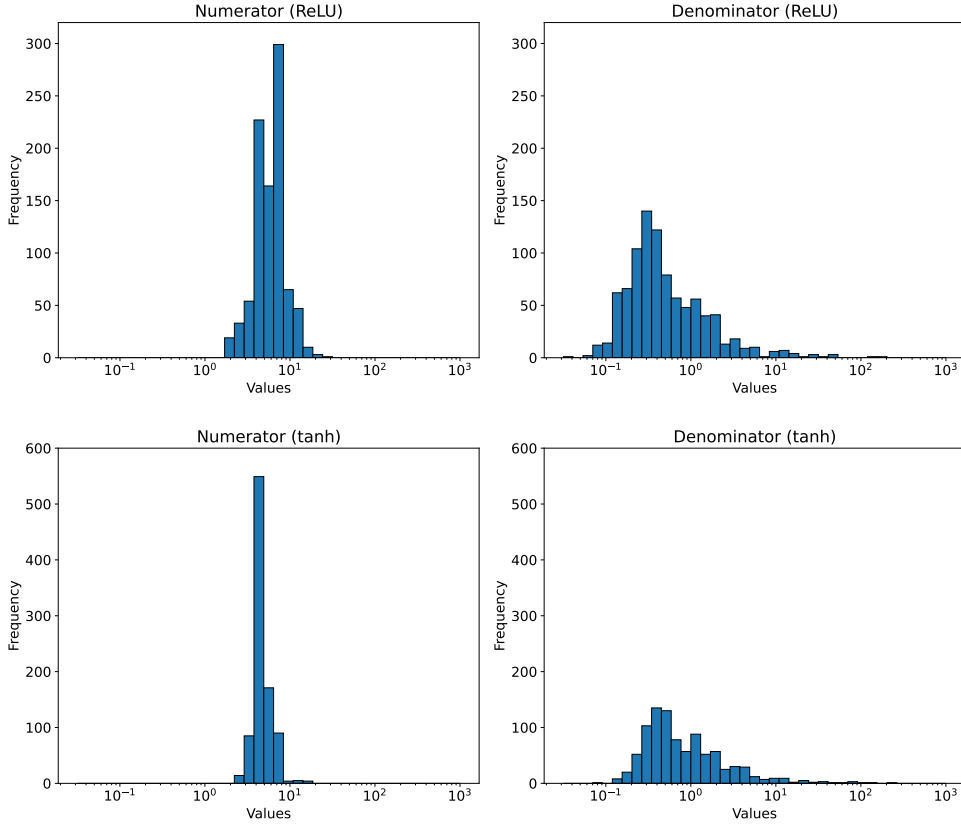
FIG. 3.2. *Distribution of the components of the numerator $|W_\ell||h_{\ell-1}|$ (left) and of the denominator $|W_\ell h_{\ell-1}|$ (right) of κ_{v_ℓ} computed in FP8, for a three-layer network trained on the MNIST dataset with ReLU (top) and tanh (bottom) activations.*

reasonable estimate of its order of magnitude. In particular, for the ReLU function, the vast majority of the zero values (corresponding to negative components of v_ℓ) in FP32 are correctly identified as zeros in FP8 also. There are a few outliers, in both directions: some FP32 zeros become nonzeros in FP8 (top left blue outliers), and may be needlessly recomputed; conversely, some FP32 nonzeros become zeros in FP8 (bottom right blue outliers), and will be kept in low precision even though they should be recomputed. These outliers represent a very small percentage of the components and we may expect them not to have a significant impact on the inference accuracy. Therefore, in the sequel, we use the low precision u_{low} to compute the condition numbers κ_ℓ.

Having computed a low precision v_ℓ, estimating $\kappa_\phi(v_\ell)$ is straightforward: it suffices to compute $\kappa_\phi(v_\ell) = |v_\ell \circ \phi'_\ell(v_\ell) \oslash \phi_\ell(v_\ell)|$ in precision u_{low}. Note that this formula involves computing $h_\ell = \phi_\ell(v_\ell)$ in precision u_{low}; the output h_ℓ of the ℓth layer in low precision is thus computed for free as part of this estimation; only the components of h_ℓ needing a higher precision will need to be recomputed. Unfortunately, estimating $\kappa_{v_\ell} = (|W_\ell||h_{\ell-1}|) \oslash |W_\ell h_{\ell-1}|$ is still too expensive, because of the expensive computation required by the numerator. Indeed, computing this numerator for all ℓ would cost the same as a full forward pass, since we need to compute the matrix–vector products $|W_\ell||h_{\ell-1}|$ for all layers.
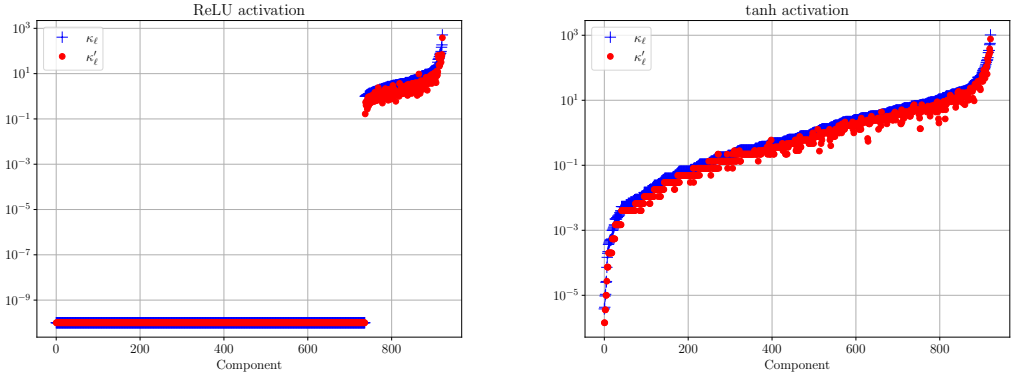
FIG. 3.3. *Comparison of the condition number $\kappa_\ell = \kappa_\phi(v_\ell) \circ \kappa_{v_\ell}$ and its proposed approximation $\kappa_\ell' = \kappa_\phi \circ \frac{c}{|W_\ell h_{\ell-1}|}$ (with $c = 3$), both computed in FP8, for a three-layer network trained on the MNIST dataset with ReLU (left) and tanh (right) activations.*

We can however avoid this computation, thanks to the key observation that the variations in magnitude of κ_{v_ℓ} are mostly due to variations of the denominator. We illustrate this in Figure 3.2, which reports the distribution of the numerator and the denominator in κ_{v_ℓ} for a three-layer network trained on MNIST. For both the ReLU (top) and the tanh (bottom) functions, the denominator (right) has a much larger dynamic range than the numerator (left). As a consequence, it seems reasonable to approximate the numerator by a fixed constant c.

Figure 3.3 confirms that the approximation $\kappa_{v_\ell} \approx \kappa_{v_\ell}' := \mathbf{1} \oslash |v_\ell|$ is reasonable. Note that there is no need to tune this constant c, because it can be directly integrated in the criterion based on τ: checking whether $c\kappa_{\phi_\ell}(v_\ell) \oslash |W_\ell h_{\ell-1}| \leq \tau$ is equivalent to checking whether $\kappa_{\phi_\ell}(v_\ell) \oslash |W_\ell h_{\ell-1}| \leq \tau'$ with $\tau' = \tau/c$. Thus the tolerance τ is the only hyperparameter that needs tuning.

3.3. *The algorithm*

The successive approximations introduced above lead to a practical criterion for a mixed precision inference evaluation strategy. We summarize the proposed approach in Algorithm 3.1.

As mentioned previously, in order for the algorithm to be efficient, the percentage of components that need to be recomputed in high precision must be small. We now quantify this statement more precisely by using the following cost model. We only consider the cost of the matrix–vector products $W_\ell h_{\ell-1}$. These require $O(n_\ell n_{\ell-1})$ floating-point operations, whereas the remaining steps of the algorithm (which essentially consist of the evaluation of the activation functions and the estimation of the condition numbers) only require $O(n_\ell)$ operations/function evaluations. Therefore for large-scale networks we may reasonably assume that the cost of the matrix–vector products will dominate—note that this specifically assumes multilayer perceptron networks; see section 5 for a discussion on the extension to convolutional networks.

Let us thus focus on the matrix–vector products. Let c_{low} be the cost of performing all the matrix–vector products (across all layers) in uniform precision u_{low}, and let c_{high} be the corresponding cost when using uniform precision u_{high} instead. Let $\rho \in [0, 1]$ be the fraction of components—and thus of inner products—that need to be recomputed in precision u_{high}. Then the cost of the mixed precision

---

Algorithm 3.1 Neural network inference with mixed precision accumulation

---

Input: $W_1, \ldots, W_L$, the weight matrices; $h_0 = x$, the input vector; τ, a tolerance controlling the precision choice; $u_{\text{low}}, u_{\text{high}}$, the precisions.
Output: h_L, the output of the network.
1: **for** $\ell = 1, \ldots, L$ **do**
2:     Compute $v_\ell = W_\ell h_{\ell-1}$ in precision u_{low}.
3:     Compute $h_\ell = \phi_\ell(v_\ell)$ in precision u_{low}.
4:     Compute $\kappa_{\phi_\ell}(v_\ell) = |v_\ell \circ \phi'_\ell(v_\ell)| \oslash |\phi_\ell(v_\ell)|$ in precision u_{low}.
5:     Compute $\kappa_\ell = \kappa_{\phi_\ell} \oslash |v_\ell|$ in precision u_{low}.
6:     **for** every component $(\kappa_\ell)_i$ **do**
7:         **if** $(\kappa_\ell)_i > \tau$ **then**
8:             Recompute $(v_\ell)_i = (W_\ell h_{\ell-1})_i$ in precision u_{high}.
9:             Recompute $(h_\ell)_i = \phi_\ell((v_\ell)_i)$ in precision u_{high}.
10:           Requantize $(h_\ell)_i$ back to precision u_{low}.
11:         **end if**
12:     **end for**
13: **end for**

Algorithm 3.1 is

$$c_{\text{mixed}} = c_{\text{low}} + \rho c_{\text{high}} = \left(\frac{c_{\text{low}}}{c_{\text{high}}} + \rho \right) c_{\text{high}}, \tag{3.1}$$

It is thus important to note that while we naturally have $c_{\text{low}} \le c_{\text{mixed}}$, we cannot guarantee in general that $c_{\text{mixed}} \le c_{\text{high}}$: for this to hold, we must have the condition $c_{\text{low}}/c_{\text{high}} + \rho < 1$. In other words, the mixed precision cost will be less than the high precision one if the costs ratio between the low and high precision is sufficiently small, and the fraction of components that need to be recomputed in high precision is also sufficiently small.

Remark 3.1 *Algorithm 3.1 can easily be extended to use more than two precisions. Indeed, given a list of precisions with unit roundoffs $u_1 > \ldots > u_p$, we can first compute h_ℓ in precision u_1 and check the components of κ_ℓ against a list of tolerances $\tau_1 < \ldots < \tau_{p-1} < \tau_p := \infty$. Components $(\kappa_\ell)_i \in (\tau_j, \tau_{j+1}]$ are then recomputed in precision u_j, for $j = 1: p-1$. The cost model (3.1) then becomes*

$$c_{\text{mixed}} = c_1 + \sum_{j=2}^{p} \rho_j c_j,$$

where c_j is the cost of computing an MMA in precision u_j and ρ_j is the fraction of components that are recomputed in precision u_j.

4. Numerical experiments

In this section we experimentally assess the potential of the mixed precision strategy introduced in Algorithm 3.1.

Experimental setting and description of the figures. We consider multilayer perceptron networks [23] with 3, 5, or 8 layers (including both the hidden and input/output layers), with either ReLU or tanh

activation functions. The weight matrices for an L-layer network have dimensions 784×784 for the first $L-2$ layers, 128×784 for layer $L-1$ and 10×128 for layer L.

Our experiments use floating-point arithmetic, with two different formats: the FP8-E4M3 format [22], an 8-bit format with 4 bits dedicated to the exponent and 3 bits to the mantissa, and the IEEE-754 FP16 format [17], a 16-bit format with 5 bits dedicated to the exponent and 10 bits to the mantissa. Hereinafter, we denote these two formats simply as FP8 and FP16, respectively. We leverage the mptorch [28] Python library to faithfully simulate reduced precision computations in FP8.

For all experiments, the neural networks considered are pre-trained on the MNIST and Fashion MNIST datasets using IEEE-754 FP32 (single precision) arithmetic and a quantization-aware training approach [24, sect. 4] where the weights are quantized to the target FP8 format.

We consider and compare three accumulation strategies in performing feed-forward computation on the chosen networks: two uniform precision variants, which use the same accumulation precision (either FP8 or FP16) across all components, and our mixed precision variant (Algorithm 3.1), which uses FP8 as the low precision u_{low} and FP16 as the high precision u_{high}.

On most hardware, we can expect FP8 arithmetic to be twice as fast as FP16 arithmetic. Thus, in our cost model, we assume $c_{\text{low}}/c_{\text{high}} = 0.5$. Then (3.1) yields

$$c_{\text{mixed}} = (0.5 + \rho)c_{\text{high}} \qquad (4.1)$$

where $\rho \in [0,1]$ is the fraction of inner products that must be recomputed in FP16. Based on (4.1) we can expect that if $\rho < 0.5$, the cost of the mixed precision FP8/FP16 method will be lower than that of the uniform FP16 one. For each network type and each precision configuration variant, we perform inference on 10,000 different test inputs and report the resulting test accuracy (that is, the percentage of inputs correctly classified).

The results are presented in Figure 4.1 for ReLU activation functions and in Figure 4.2 for tanh. In each figure, the top, middle, and bottom plots correspond to networks with 3, 5, and 8 layers, respectively. The left and right plots correspond to the MNIST and Fashion MNIST datasets, respectively. Each individual plot shows the test accuracy on the x-axis and the fraction ρ of inner products (re)computed in FP16 on the y-axis, for each of the three precision configurations: uniform FP8 (a single triangle marker, always found at $y = 0$), uniform FP16 (a single star marker, always found at $y = 1$), and the mixed precision Algorithm 3.1 with various values for the tolerance τ (blue line).

The figures show that different precision configurations achieve different cost–accuracy tradeoffs. Without surprise, the uniform FP16 variant is always more accurate than the FP8 one. As for the mixed precision variant, we see that decreasing the tolerance τ increases the accuracy but also increases the fraction of inner products that need to be recomputed in FP16. Based on the cost model (4.1), we also plot a dashed line at $\rho = 0.5$, the maximum value for which the cost of the mixed precision algorithm remains less than the uniform FP16 one. Hence blue points below that dashed line are potentially of interest.

Discussion of the results when using ReLU activation functions. The results of these experiments are reported in Figure 4.1. For the ReLU function for any choice of the tolerance τ, only a tiny fraction of the inner products need to be recomputed in FP16. This is due to the fact that $\kappa_\phi(x) = 0$ if $x < 0$, meaning that any inner product whose result is negative will systematically be kept in low precision regardless of τ. As it turns out, the percentage of negative inner products, and thus of zero condition numbers, is extremely large. Table 4.1 summarizes these percentages (averaged over all inputs) for the different types of networks; they are very large regardless of the dataset or of the number of layers, exceeding 75% in all cases. This explains why, in the left plots, the blue points corresponding to the mixed

TABLE 4.1 *Average percentage of zero values in the condition number of ReLU activations for multilayer perceptron networks with 3, 5, or 8 layers trained on the MNIST and Fashion MNIST datasets, using FP8 arithmetic.*

Multilayer Perceptron Configuration	MNIST	Fashion MNIST
3 layers	84%	90%
5 layers	80%	85%
8 layers	77%	80%

precision configuration never exceed a fraction of $\rho = 0.25$ inner products recomputed in FP16. Thus, we are far below the $\rho = 0.5$ limit and we can expect the mixed precision variant to be significantly faster than the uniform FP16 one.

Despite the large number of operations performed in FP8, the mixed precision variant always achieves a better accuracy than the uniform FP8 variant. More importantly, for a sufficiently small tolerance τ, its accuracy matches that of the uniform FP16 variant. Thus, the mixed precision variant is *faster yet equally as accurate* as the uniform FP16 variant. It is interesting to note that as we increase τ, the fraction ρ of inner products needing to be recomputed in FP16 does slightly decrease, from roughly 0.2 to 0.1. Since for ReLU κ_{ϕ_ℓ} is either 0 or 1, this behavior is explained by the variations in the components of κ_{v_ℓ}. Specifically, for very small values of τ, components with $\kappa_{\phi_\ell} = 1$ will always be recomputed in FP16. As we increase τ, some of these components may be kept in FP8 if κ_{v_ℓ} is small enough, further reducing the fraction of FP16 computations. However, the figure shows that the test accuracy quickly degrades when doing so, for a cost reduction that is not that significant. Therefore, these experiments suggest that for ReLU activations, a good rule of thumb is to recompute in FP16 all positive inner products (for which $\kappa_{\phi_\ell} = 1$).

All these observations hold consistently for all the tested networks, even as we increase the number of layers, both for the MNIST and Fashion MNIST datasets.

Discussion of the results when using tanh ***activation functions.*** For the tanh function, the situation is quite different, as shown in Figure 4.2. The fraction ρ of inner products needing to be recomputed in FP16 quickly increases as τ decreases, so that not all mixed precision configurations are interesting. Indeed, in view of (4.1), all choices of τ that demand to recompute more than $\rho = 0.5$ of the inner products (blue points above the dashed line) should be discarded, since they are more expensive than the uniform FP16 variant, and yet achieve a lower test accuracy as shown in the figure. However, some choices of τ still provide an interesting compromise between accuracy and cost. The largest values of τ (for example, $\tau = 5$) often still allow for a slight improvement of the accuracy with respect to uniform FP8, which comes almost for free since $\rho \approx 0.9$ in these cases. Alternatively, more intermediate values of τ (for example, $\tau = 1$) can achieve much more significant accuracy improvements (without, however, reaching the same accuracy as FP16), for a cost that is in between that of the uniform FP8 and FP16 variants (for example, $\rho \approx 0.3$, which corresponds to a 20% cost reduction with respect to uniform FP16 in view of (4.1)).

Overall, these experimental results support the conclusions of our analysis, confirming that it is indeed meaningful to compute different components of the layers in different precisions, and highlight the potential of the proposed Algorithm 3.1 to improve the cost–accuracy tradeoff, particularly in the case of ReLU activations.
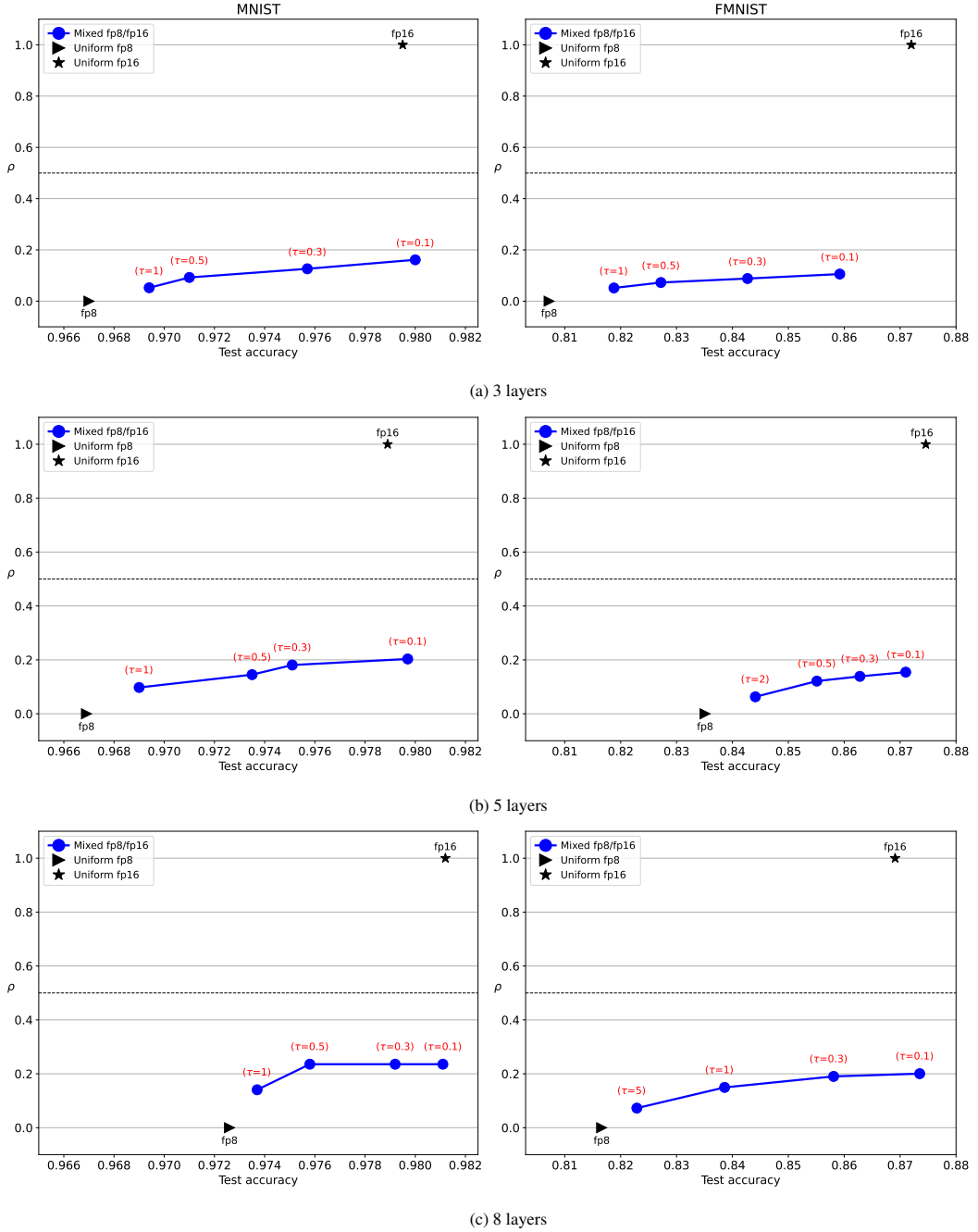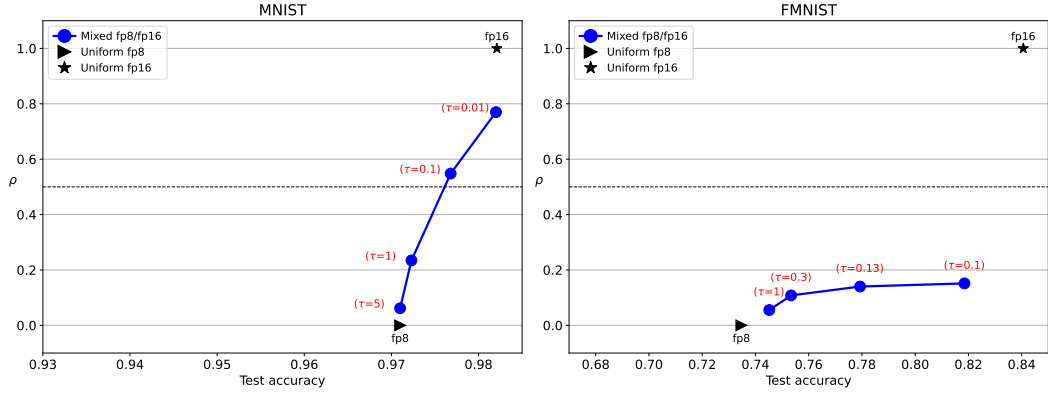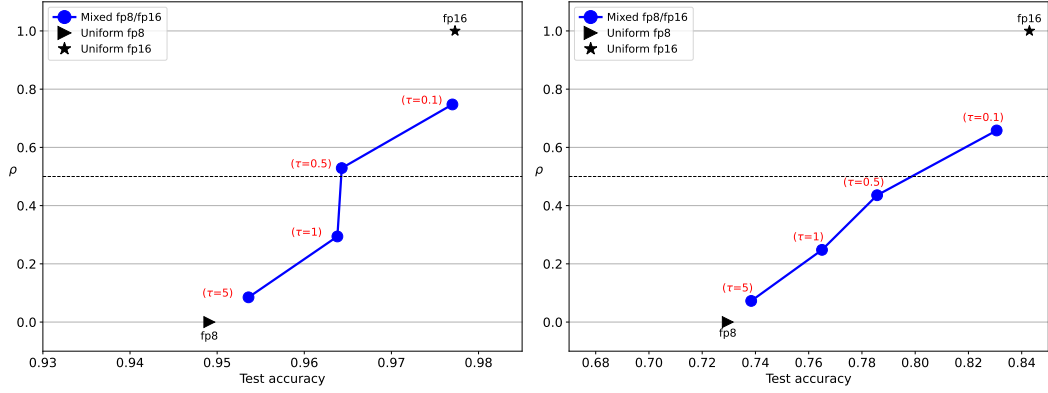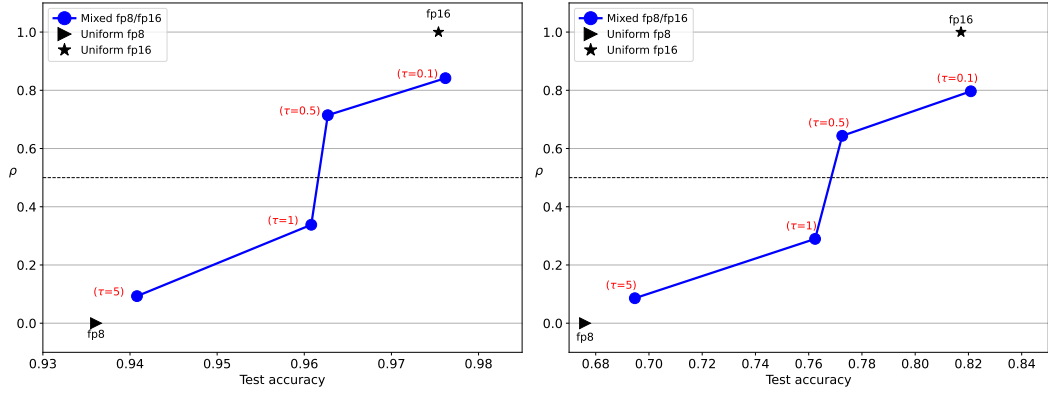
(a) 3 layers

(b) 5 layers

(c) 8 layers

FIG. 4.1. *Cost–accuracy tradeoff achieved by different precision configurations on the MNIST (left) and Fashion MNIST (right) datasets, for multilayer perceptron networks with 3 (top), 5 (middle), or 8 (bottom) layers, using ReLU activation. The x-axis plots the test accuracy of the inference on the 10,000 samples of the dataset; the y-axis plots the fraction ρ of inner products (re)computed in FP16. For the mixed precision configuration (Algorithm 3.1), each point corresponds to a different value of the tolerance τ as indicated.*

(a) 3 layers

(b) 5 layers

(c) 8 layers

FIG. 4.2. Same as Figure 4.1 but with tanh activations.

5. Conclusion

We have considered the problem of using mixed precision accumulation in the matrix–multiply accumulate operations for neural network inference. In order to do so, we investigated the propagation of errors in the inference, based on a generic error model that applies in particular to floating-point arithmetic. Specifically, we have carried out a componentwise forward error analysis, whose main conclusion is reported in Theorem 2.4. This key result shows that the errors incurred in each inner product of each layer are proportional to the condition number of the inner product and to the condition number of the activation functions. Therefore our analysis suggests (see Corollary 2.5) to choose the precision of each inner product to be inversely proportional to this product of condition numbers.

We have leveraged this insight by developing an inference algorithm with mixed precision accumulation. We introduced some approximations in order to cheaply estimate the condition numbers, leading to the practical approach outlined in Algorithm 3.1 and illustrated in Figure 1.1. We have validated the soundness and potential of this approach experimentally on multilayer perceptrons networks with ReLU and tanh activations. Our experimental results indeed show that the proposed mixed precision approach can significantly improve the cost–accuracy tradeoff: in most cases, it is more accurate than the low precision baseline (FP8 in our tests) and less expensive than the high precision baseline (FP16 in our tests).

The analysis is general enough to cover various network architectures. We have focused our experiments on the multilayer perceptron one, but the approach could be adapted to convolutional networks. However, an analysis taking into account the specific structure of such networks should lead to sharper bounds and is left for future work.

Acknowledgments

REFERENCES

1. *CUDA PTX ISA*. NVIDIA, May 2024. Release 8.5.
2. T. Beuzeville, A. Buttari, S. Gratton, and T. Mary. Deterministic and probabilistic backward error analysis of neural networks in floating-point arithmetic. HAL EPrint hal-04663142.
3. P. Blanchard, N. J. Higham, F. Lopez, T. Mary, and S. Pranesh. Mixed Precision Block Fused Multiply-Add: Error Analysis and Application to GPU Tensor Cores. *SIAM J. Sci. Comput.*, 42(3):C124–C141, 2020.
4. P. Blanchard, N. J. Higham, and T. Mary. A Class of Fast and Accurate Summation Algorithms. *SIAM J. Sci. Comput.*, 42(3):A1541–1557, 2020.
5. I. Colbert, F. Grob, G. Franco, J. Zhang, and R. Saab. Accumulator-aware post-training quantization. *arXiv preprint arXiv:2409.17092*, 2024.
6. I. Colbert, A. Pappalardo, and J. Petri-Koenig. Quantized neural networks for low-precision accumulation with guaranteed overflow avoidance. *arXiv preprint arXiv:2301.13376*, 2023.
7. M. P. Connolly, N. J. Higham, and T. Mary. Stochastic Rounding and its Probabilistic Backward Error Analysis. *SIAM J. Sci. Comput.*, 43(1):A566–A585, 2021.
8. M. Croci, M. Fasi, N. J. Higham, T. Mary, and M. Mikaitis. Stochastic rounding: Implementation, error analysis and applications. *Roy. Soc. Open Sci.*, 9(3):1–25, 2022.
9. B. De Bruin, Z. Zivkovic, and H. Corporaal. Quantization of deep neural networks for accumulator-constrained processors. *Microprocessors and microsystems*, 72:102872, 2020.

10. Z. Dong, Z. Yao, D. Arfeen, A. Gholami, M. W. Mahoney, and K. Keutzer. HAWQ-v2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems*, 33: 18518–18529, 2020.

11. Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer. HAWQ: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pages 293–302.

12. E.-M. El Arar, M. Fasi, S.-I. Filip, and M. Mikaitis. Probabilistic error analysis of limited-precision stochastic rounding, 2025.

13. A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-power computer vision*, Chapman and Hall/CRC, 2022, pages 291–326.

14. C. Gong, Z. Jiang, D. Wang, Y. Lin, Q. Liu, and D. Z. Pan. Mixed precision neural architecture search for energy efficient deep learning. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2019, pages 1–7.

15. S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *International conference on machine learning*, PMLR, 2015, pages 1737–1746.

16. N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Second edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002. xxx+680 pp. ISBN 0-89871-521-0.

17. *IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2019 (revision of IEEE Std 754-2008)*. Institute of Electrical and Electronics Engineers, Piscataway, NJ, USA, July 2019. 82 pp. ISBN 978-0-7381-5752-8.

18. C.-P. Jeannerod and S. M. Rump. Improved error bounds for inner products in floating-point arithmetic. *SIAM J. Matrix Anal. Appl.*, 34(2):338–344, 2013.

19. N. P. Jouppi and et all. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pages 1–12.

20. D. Lin, S. Talathi, and S. Annapureddy. Fixed point quantization of deep convolutional networks. In *International conference on machine learning*, PMLR, 2016, pages 2849–2858.

21. T. Mary and M. Mikaitis. Error analysis of matrix multiplication with narrow range floating-point arithmetic. HAL EPrint hal-04671474.

22. P. Micikevicius, S. Oberman, P. Dubey, M. Cornea, A. Rodriguez, I. Bratt, R. Grisenthwaite, N. Jouppi, C. Chou, A. Huffman, M. Schulte, R. Wittig, D. Jani, and S. Deng. OCP 8-bit floating point specification (OFP8), June 2023. Version 1.0. 16 pp.

23. F. Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5):183–197, 1991.

24. M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

25. R. Ni, H. Chu, C. F. O., P. Chiang, C. Studer, and T. Goldstein. Wrapnet: Neural net inference with ultra-low-precision arithmetic. In *International Conference on Learning Representations ICLR 2021*, OpenReview, 2021.

26. C. Sakr, N. Wang, C. Chen, J. Choi, A. Agrawal, N. Shanbhag, and K. Gopalakrishnan. Accumulation bit-width scaling for ultra-low precision training of deep networks. *arXiv preprint arXiv:1901.06588*, 2019.

27. M. A. Talib, S. Majzoub, Q. Nasir, and D. Jamal. A systematic literature review on hardware implementation of artificial intelligence algorithms. *J Supercomput*, 77:1897–1938, 2021.

28. M. Tatsumi, Y. Xie, C. White, S.-I. Filip, O. Sentieys, and G. Lemieux. MPTorch and MPArchimedes: Open Source Frameworks to Explore Custom Mixed- Precision Operations for DNN Training on Edge Devices. In *ROAD4NN 2021 - 2nd ROAD4NN Workshop: Research Open Automatic Design for Neural Networks*, San Francisco, United States, December 2021.

29. S. Uhlich, L. Mauch, F. Cardinaux, K. Yoshiyama, J. A. Garcia, S. Tiedemann, T. Kemp, and A. Nakamura. Mixed precision DNNs: All you need is a good parametrization. *arXiv preprint arXiv:1905.11452*, 2019.

30. K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han. HAQ: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pages 8612–8620.

31. N. Wang, J. Choi, D. Brand, C. Chen, and K. Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, Red Hook, NY, USA, 2018, pages 7686–7695. Curran Associates Inc.

32. H. Xie, Y. Song, L. Cai, and M. Li. Overflow aware quantization: Accelerating neural network inference by low-bit multiply-accumulate operations. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2021, pages 868–875.

33. H. Yang, L. Duan, Y. Chen, and H. Li. BSQ: Exploring bit-level sparsity for mixed-precision neural network quantization. 2021.

34. Z. Yao, Z. Dong, Z. Zheng, A. Gholami, J. Yu, E. Tan, L. Wang, Q. Huang, Y. Wang, M. W. Mahoney, and K. Keutzer. HAWQ-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*, PMLR, 2021, pages 11875–11886.